**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу Кафедра штучного інтелекту**

**Звіт**

**про виконання домашньої роботи №2**

**з дисципліни «Технології розподілених систем і паралельних обчислень»**

Виконав:

студент II курсу, групи КІ-31

Микитин Владислав Олександрович

Прийняв:

Кот Анатолій Тарасович

Київ – 2024

**Мета роботи**:

* закріплення знань про метод Монте-Карло для наближеного обчислення числа π;
* набуття практичних навичок реалізації багатопотокових програм у Python;
* дослідження впливу кількості потоків на час виконання обчислень;
* аналіз ефективності багатопотокового підходу в умовах використання інтерпретованої мови Python та впливу GIL;
* формування навичок збору експериментальних даних, побудови таблиць і звітів для порівняння продуктивності.

**Теоретичні відомості**:

Метод Монте-Карло для оцінки π (класичний підхід):

* Беремо квадрант одиничного круга (коротко — вписаний чверть-круг у квадрат [0,1]×[0,1]).
* Генеруємо N випадкових точок (x,y) рівномірно в квадраті.
* Обчислюємо кількість точок, що потрапили в чверть круга: x^2 + y^2 <= 1.
* Ймовірність потрапляння в круг ≈ π/4. Тому оцінка π:

Статистичні характеристики оцінки:

* Нехай p = π/4 ≈ 0.7853981634 — ймовірність «усередині». Для пропорції дисперсія ≈ p(1−p)/N.
* Для N = 1{,}000{,}000 математично очікувана стандартна помилка оцінки π:

Отже, випадкові відхилення порядку ~0.0016 є нормальними для одного запуску з N=1e6.

Про багатопоточність у Python:

Модуль threading використовує один процес та GIL (Global Interpreter Lock) у CPython. Для CPU-bound обчислень GIL часто заважає: кількість активних потоків, що виконують обчислення одночасно в Python-інтерпретаторі, зазвичай не дає лінійного прискорення. Додаткові фактори: накладні витрати на створення та планування потоків, блокування генератора випадкових чисел (global random), контекстні переключення тощо.

**Завдання:**

Обчислити число π методом Монте-Карло на 1\_000\_000 випадкових точок у двох варіантах:

* Головний потік (single-thread).
* Паралельно потоками: 2, 4, 8, 16, 32, 64 (ті ж 1\_000\_000 точок, розбиті порівну між потоками).

Заміряти час обчислення для кожного варіанту. Не використовувати жодних механізмів синхронізації, крім join для очікування завершення потоків. Зберегти / продемонструвати результати у вигляді таблиці (та підготувати звіт для Git).

**Хід виконання завдання:**

1. Реалізовано функцію, яка генерує n випадкових точок і підраховує, скільки з них в чверть-кругу.
2. Написано дві обгортки:
   1. single\_thread\_pi(n\_points) — виклик функції у головному потоці.
   2. multi\_thread\_pi(n\_points, n\_threads) — розбиття n\_points на n\_threads (по n\_points // n\_threads), запуск потоків; після join збір результатів і обчислення π.
3. Для кожної конфігурації (1,2,4,8,16,32,64 потоків) проведено по одному запуску і зафіксовано час виконання (time.time()).
4. Результати збережено у pandas.DataFrame (та CSV) і виведено у таблиці.

**Лістинг програми (Jupyter Notebook — основні клітинки):**

# Notebook cell 1: імпорти

import threading

import random

import time

import pandas as pd

import math

# Notebook cell 2: функція Монте-Карло (використовувалась для оцінки)

def monte\_carlo\_pi(n\_points, results, idx):

inside\_circle = 0

for \_ in range(n\_points):

x, y = random.random(), random.random()

if x\*x + y\*y <= 1.0:

inside\_circle += 1

results[idx] = inside\_circle

# Notebook cell 3: single-thread run

def single\_thread\_pi(n\_points=1\_000\_000):

start = time.time()

results = [0]

monte\_carlo\_pi(n\_points, results, 0)

pi\_estimate = 4 \* results[0] / n\_points

elapsed = time.time() - start

return pi\_estimate, elapsed

# Notebook cell 4: multi-thread run (лише join для синхронізації)

def multi\_thread\_pi(n\_points=1\_000\_000, n\_threads=4):

points\_per\_thread = n\_points // n\_threads

threads = []

results = [0] \* n\_threads

start = time.time()

for i in range(n\_threads):

t = threading.Thread(target=monte\_carlo\_pi, args=(points\_per\_thread, results, i))

threads.append(t)

t.start()

for t in threads:

t.join()

total\_inside = sum(results)

pi\_estimate = 4 \* total\_inside / n\_points

elapsed = time.time() - start

return pi\_estimate, elapsed

# Notebook cell 5: експеримент і збереження результатів

n\_points = 1\_000\_000

thread\_counts = [1, 2, 4, 8, 16, 32, 64]

results = []

for n\_threads in thread\_counts:

if n\_threads == 1:

pi, elapsed = single\_thread\_pi(n\_points)

else:

pi, elapsed = multi\_thread\_pi(n\_points, n\_threads)

results.append({"Threads": n\_threads, "Pi Estimate": pi, "Time (s)": elapsed})

df = pd.DataFrame(results)

print(df)

df.to\_csv("results.csv", index=False)

Примітка: у наведеному лістингу використовується глобальний генератор random.random() (стандартний модуль random). Це робочий варіант і відповідає умові «лише join як синхронізація». Однак варто знати, що глобальний генератор має внутрішні блокування, що може впливати на швидкодію при багатьох потоках.

**Аналіз алгоритму програми:**

1. Структура:
   1. Основна частина — підрахунок точок, що потрапили всередину кола (monte\_carlo\_pi).
   2. Однопотокове обчислення (single\_thread\_pi) — виконує всі N точок у головному потоці.
   3. Багатопотокове обчислення (multi\_thread\_pi) — ділить N на T потоків, кожен підраховує свою частину, результати сумуються.
2. Складність:
   1. Часова: O(N) — кількість операцій пропорційна N, незалежно від потоків.
   2. Пам’ять: O(T) для масиву результатів, мінімальні константні витрати.
3. Синхронізація:
   1. Використовується лише join для очікування завершення потоків.
   2. Кожен потік пише у свій елемент масиву results, тому перехресного доступу немає.
4. Особливості роботи потоків у Python:
   1. GIL блокує одночасне виконання Python-байткоду, тому для CPU-bound задач багатопоточність не дає лінійного прискорення.
   2. Використання глобального генератора випадкових чисел (random.random()) може створювати невеликі додаткові блокування.

**Демонстрація результатів:**

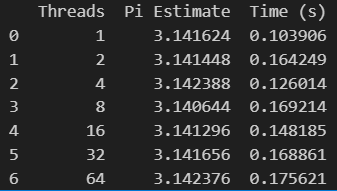
****

Рисунок 1 – зображення результатів виконання програми

**Аналіз результатів:**

1. Точність оцінки π:
   1. Усі значення близькі до 3.14159265.
   2. Абсолютна похибка ≈ 0.001, що відповідає теоретичній стандартній помилці для 1 000 000 точок.
2. Час виконання:
   1. Однопотоковий варіант найшвидший (0.1039 с).
   2. Збільшення числа потоків не дає прискорення; час навіть зростає через накладні витрати на створення потоків і GIL.
3. Висновок:
   1. У Python threading неефективний для CPU-bound обчислень.
   2. Додаткові потоки збільшують накладні витрати, не скорочуючи час виконання.

**Висновки:**

1. Метод Монте-Карло на 1 млн точок забезпечує точну оцінку числа π.
2. Однопотокове виконання у CPython є швидшим за багатопоточне для CPU-bound задач.
3. Для реального прискорення багатопоточних обчислень рекомендовано:
   1. використовувати multiprocessing;
   2. векторизувати обчислення через NumPy;
   3. або застосовувати C/Numba/Cython для оптимізації.
4. Завдання дало практику:
   1. роботи з потоками у Python,
   2. використання join для синхронізації,
   3. збору та аналізу експериментальних даних у Jupyter Notebook.